DCP Data Service (DDS) Protocol Specification

Revision 2.1

4/04/2003

Prepared For

U.S. Geological Survey,
Water Resources Division

U.S. Army Corps of Engineers

National Oceanic and
Atmospheric Administration

Prepared by

Ilex Engineering, Inc.
Web: www.ilexeng.com
Email: info@ilexeng.com

Table of Contents

1 Introduction

DDS stands for “DCP Data Service”. It is a client/server protocol for efficiently transferring DCP data over a network. DDS is in wide use among agencies that use the GOES (Geosynchronous Operational Environmental Satellite) DCS (Data Collection System).

This document provides a description of DDS and its history. It also defines the client server protocol in detail.

1.1 History of DDS

There are four versions of DDS defined.

DDS Protocol Version 1:

DDS was originally developed by Integral Systems, Inc., as part of the DOMSAT Receive Station product. The DOMSAT Receive Station collected satellite data and stored it in a circular file on the hard disk. Clients could connect using DDS and retrieve any subset of data, either historical or in real-time.

Ilex Engineering, Inc. (henceforth "Ilex") purchased the DOMSAT Receive Station copyrights in September of 2000. Ilex currently maintains the DDS implementation in DOMSAT Receive Stations.

USGS, BLM, and other organizations coded their applications to act as DDS clients, pulling data from a DOMSAT system in real-time.

DDS Protocol Version 2:

In 2000, Ilex produced a Java implementation of DDS for use in the open source LRGS (Local Readout Ground Station) DOMSAT receiver. This implementation enhances the original by allowing for the transfer of network list files. This is an important capability because it makes a client more independent from the server. The client can start a session by downloading all needed network lists. Before, the client would have to rely on persistent lists that were pre-loaded on the server.

DDS Protocol Version 3:

For LRGS Release 3.3, Ilex added a password-protection mechanism to DDS. This work was done under contract to the USGS. The mechanism uses a non-reversible hash of the passwords to prevent detection of passwords by monitoring network traffic. This document provides the details on the password exchange when a client establishes a connection.

DDS Protocol Version 4 and 5:

Ilex participated in an effort to build a replacement for the central DCP message processing system in Wallops, VA. This work was done for NOAA/NESDIS (National Oceanic and Atmospheric Administration, National Environmental Satellite Data Information Service). The new system, called "DAPS-II" will incorporate DDS as a primary mechanism for distributing DCP data to the user community. For DAPS-II, an enhancement has been added to DDS to improve performance, especially when used over a wide area network.

Protocol Version 5 is identical to 4. The additional version is added because initial server implementations of the 'message-block' feature were not reliable on some OS platforms. This feature was subsequently tuned and tested on a variety of platforms. Current clients are recommended to not use the message-block feature unless the server supports protocol version 5 or higher.

Successive versions are additive. None of the original features have been deprecated.

1.2 RFC 2119 Conformance

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in IETF RFC 2119, which can be found at:

http://www.ietf.org/rfc/rfc2119.txt?number=2119

1.3 BNF Notation

This document uses BNF (Backus Naur Form) to define the syntax of messages sent between client and server. The following conventions are used:

Notation	***Meaning***
`::=`	Is defined as
`'literal'`	A literal string is enclosed in single quotation marks
`nonterminal`	Non-terminal symbols are not enclosed in quotation marks. It must be recursively defined elsewhere.
`one | two`	Pipe symbol means ‘or’. This rule means “one or two”.
`{ rule }`	Curly brackets mean zero or more repetitions of rule.
`[ optional ]`	Rules in square brackets are optional.
`DIGIT`	Any ASCII digit 0 through 9
`CRLF`	ASCII Carriage Return followed by Line Feed
`SP`	ASCII Space Character
`STRING`	Any sequence of printable ASCII characters except CRLF. May contain space or tab characters.
`OCTET_STRING`	Any sequence of 8-bit binary octet values. This is only used for transferring DCP message data.
`( group of symbols )`	Parentheses used for grouping within rules.
`# comment`	Characters after an un-quoted pound sign are comments.

Table 1-1: BNF Conventions Used in this Document.

Some requests and responses contain a time stamp. All time stamps MUST be in UTC and SHALL be formatted as follows:

```
time ::= YYDDDHHMMSS
```

‘YY’ is the last two digits of the year.
‘DDD’ is the Julian day of the year (January 1 == day 1)
‘HHMMSS’ is the UTC hour, minute, and second of the day.

Integers are made up of at least one digit:

```
integer ::= DIGIT { DIGIT }
```

Hex numbers are represented by <hexstring>:

```
hexstring ::= hexdigit { hexdigit }
hexdigit :: DIGIT |
            ‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ |
            ‘A’ | ‘B’ | ‘C’ | ‘D’ | ‘E’ | ‘F’ |
```

A “NAME” is an alphanumeric string that contains no whitespace. It must begin with a letter.

```
NAME ::= letter { letter | digit | underscore }
underscore ::= '_'
```

A special identifier ‘empty’ is occasionally used to explicitly indicate a field that contains no data (i.e. zero length)

2 DDS Protocol Messages

This section describes the general features of the protocol that are observed by all message types.

2.1 DDS Request/Response Headers

Each request and response is composed of a 10-byte header followed by a variable length body. Protocol messages are constructed as follows:

```
DdsMessage ::= header body
header ::= sync type length
sync ::= 'FAF0'    # last letter is a zero.
type ::= octet     # unique type codes defined for each message
length ::= DIGIT DIGIT DIGIT DIGIT DIGIT  # 5-digit number
body ::= OCTET_STRING
```

First 4 bytes MUST be the ASCII characters “FAF0” (the last character is a zero).

The next byte contains the message type. Type-codes for each request are described below.

The next 5 bytes is a five-digit number, zero-filled. This specifies the exact number of bytes contained in the body to follow.

All client requests and server responses MUST be valid DdsMessages, as defined above. For each request, the server MUST send a single response.

The body portion of requests and responses varies with each message type and are described in the following sections.

2.2 Normal and Error Responses

A server MUST respond to a request with either a normal response or an error response. Exactly one response MUST be returned for each request.

The body portion of an error responses MUST be formatted as follows:

```
ErrorBody ::= '?' ServerCode ',' SystemCode ',' [ explanation ]
ServerCode ::= integer
SystemCode ::= integer
explanation ::= STRING    # optional free-form ASCII string
```

The SystemCode is a Unix ‘errno’ value. This may be zero if the error was internal to the server. It will be non-zero if the problem was a system error, for example, attempting to retrieve a network list file that does not exist.

ServerCodes were originally designed for use on DOMSAT systems. Currently defined codes are shown in Table 2-1.

Several of these codes were invented to support various iterations of DOMSAT receivers and may have no meaning to other server implementations. Servers SHOULD refrain from defining new codes unless absolutely necessary.

Name	*Code*	*Description*
DSUCCESS	0	
DNOFLAG	1	Could not find start of message flag.
DDUMMY	2	Message found (and loaded) but it's a dummy.
DLONGLIST	3	Network list was too long to upload.
DARCERROR	4	Error reading archive file.
DNOCONFIG	5	Cannot attach to configuration shared memory
DNOSRCHSHM	6	Cannot attach to search shared memory
DNODIRLOCK	7	Could not get ID of directory lock semephore
DNODIRFILE	8	Could not open message directory file
DNOMSGFILE	9	Could not open message storage file
DDIRSEMERR	10	Error on directory lock semephore
DMSGTIMEOUT	11	Timeout waiting for new messages
DNONETLIST	12	Could not open network list file
DNOSRCHCRIT	13	Could not open search criteria file
DBADSINCE	14	Bad since time in search criteria file
DBADUNTIL	15	Bad until time in search criteria file
DBADNLIST	16	Bad network list in search criteria file
DBADADDR	17	Bad DCP address in search criteria file
DBADEMAIL	18	Bad electronic mail value in search criteria file
DBADRTRAN	19	Bad retransmitted value in search criteria file
DNLISTXCD	20	Number of network lists exceeded
DADDRXCD	21	Number of DCP addresses exceeded
DNOLRGSLAST	22	Could not open last read access file
DWRONGMSG	23	Message doesn't correspond with directory entry
DNOMOREPROC	24	Can't attach: No more proccesses allowed
DBADDAPSSTAT	25	Bad DAPS status specified in search criteria.
DBADTIMEOUT	26	Bad TIMEOUT value in search crit file.
DCANTIOCTL	27	Cannot ioctl() the open serial port.
DUNTILDRS	28	Specified 'until' time reached
DBADCHANNEL	29	Bad GOES channel number specified in search crit
DCANTOPENSER	30	Can't open specified serial port.
DBADDCPNAME	31	Unrecognized DCP name in search criteria
DNONAMELIST	32	Cannot attach to name list shared memory.
DIDXFILEIO	33	Index file I/O error
DNOSRCHSEM	34	Cannot attach to search semaphore
DUNTIL	35	Specified 'until' time reached
DJAVAIF	36	Error in Java - Native Interface
DNOTATTACHED	37	Not attached to LRGS native interface
DBADKEYWORD	38	Bad keyword
DPARSEERROR	39	Error parsing input file
DNONAMELISTSEM	40	Cannot attach to name list semaphore.
DBADINPUTFILE	41	Cannot open or read specified input file
DARCFILEIO	42	Archive file I/O error
DNOARCFILE	43	Archive file not opened
DICPIOCTL	44	Error on ICP188 ioctl call
DICPIOERR	45	Error on ICP188 I/O call
DINVALIDUSER	46	Invalid user name
DDDSAUTHFAILED	47	DDS Authentication Failure

Table 2-1: Currently Defined Error Codes

Table 2-2 contains the valid type-codes for DDS messages

Type-Code	***Name***	***Description***
a	IdHello	Client unauthenticated connect request. Server accept/reject response.
b	IdGoodbye	Client sends terminal handshake message before disconnect. Echoed back to client by server.
f	IdDcp	Client request for next DCP message. Server response containing error or DCP message.
g	IdCriteria	Client reads or writes search criteria on the server. The same type-code used for bidirectional transfer. See section 4 for details.
j	IdPutNetlist	Client uploads a network list to the server. Server accept/reject response.
k	IdGetNetlist	Client requests download of a network list from the server. Server response contains error or the network list.
e	IdStop	Used to abort data retrievals that may take a long time to time-out. This command is essentially a NOOP. Server echoes this message as a response to the abort.
m	IdAuthHello	Authenticated connect message containing hash of password. Server accept/reject response.
n	IdDcpBlock	Client request for next block of DCP messages. Server response containing multiple DCP messages in one DdsMessage

Table 2-2: Type-Codes used in DDS Messages.

3 Connecting and Disconnecting

This section describes how connections are established and broken.

3.1 TCP Sockets

DDS is a simple client/server protocol running over TCP sockets. The server establishes a listening socket. The client connects to the port number for this socket. A new bidirectional socket is then established for communication between client and server.

By default, the server SHOULD listen on port 16003. Older implementations used 9999. Clients and servers SHOULD be coded so that the listening port is configurable.

After establishing the socket, the first request from the client MUST be one of the two authentication mechanisms described below. Any requests sent prior to a valid Authentication Exchange MUST generate an error response from the server.

3.2 Authentication by Assertion

This message type is supported by all protocol versions. However, Version 3 (and above) servers MAY disallow authentication-by-assertion if they only want to support authenticated clients. In this case the server MUST return an error to this request.

"Authentication by Assertion" means that the client simply asserts an identity by passing a username to the server. If the username matches a valid user on the server, the connection is accepted.

Authentication by Assertion is safe under the following conditions:

- The server maintains search criteria and network list files in a temporary session-directory.
- The server places limits on the size and number of network lists to be stored.
- The client does not make assumptions about what network lists are currently available on the server (i.e. it should always upload the list at the start of each session).

The type-code for IdHello is 'a'. The body of the request and response MUST be as follows:

```
HelloRequest ::= username

username ::= NAME   # no more than 80 chars

HelloResponse ::= AcceptResponse | ErrorBody

AcceptResponse ::= username [ SP ProtocolVersion ]

ProtocolVersion ::= integer
```

The body of the request contains the user name. Older implementations padded this name to 80 characters by adding spaces to the right. Servers MUST support this.

On success, the server MUST send an AcceptResponse, containing the username, and optionally, an integer representing the highest protocol version supported by this server. If the protocol version is not present in the response, the client SHOULD assume protocol version 1.

3.3 Authenticated Connection

This message type is supported by protocol version 3 and higher. Clients MUST NOT send this message to servers with a lower protocol version. Servers SHOULD be configurable as to whether they support and/or require authenticated connections.

An ‘authenticated hello message’ has a type-code of IdAuthHello = ‘m’. The body is as follows:

```
AuthHelloBody ::= username SP time SP AuthenticatorHash
username ::= NAME
# time is UTC date/time stamp in format YYDDDHHMMSS
AuthenticatorHash ::= hexstring     # exactly 40 hex digits
# Response is as follows:
AuthHelloResp ::= AuthAcceptResp | ErrorBody
AuthAcceptResp ::= username SP time SP ProtocolVersion
ProtocolVersion ::= integer
```

Username must represent a valid user on the server. The time should be the current time in the UTC (GMT) time-zone. The server SHOULD check for the reasonableness of the time in order to prevent replay attacks. Clients and servers MUST disallow zero-length usernames or passwords.

The AuthenticatorHash is a 40-character hex representation of a 20-byte SHA hash code. Clients and servers MUST construct the hash-code as follows:

1. Construct a preliminary 20-byte hash code with no time component. The hash should be constructed from:
   - username
   - password
   - username
   - password
2. The preliminary hash represents a shared-secret that is stored on the server, and supplied by the user. Client software should query the user for the password in a secure manner.
3. The authenticator is another 20-byte SHA hash. It is constructed from:
   - username
   - preliminary hash
   - time-bytes: 4-byte integer representing time since Unix epoch, in big-endian order.
   - username
   - preliminary hash
   - time-bytes
4. Convert the 20-byte authenticator hash to a 40 byte Hexadecimal string. Use capital letters for A-F.

3.4 Disconnecting

This capability is supported by all protocol versions.

The proper way to terminate a connection is to send the IdGoodbye (type='b') message, wait for the response, and then close the socket. Clients SHOULD be coded this way.

The IdGoodbye message has an empty (zero-length) body. Upon receiving such a message, the server MUST simply echo the request back to the client.

If a client simply closes the socket, the server will most-likely detect this and close the socket properly. Sometimes, particularly over WAN connections, the server may not detect this right away. Servers SHOULD implement a timeout mechanism such that clients that have issued no requests in N seconds can be disconnected. The value of N SHOULD be configurable on the server.

4 Transferring Search Criteria to the Server

Capabilities in this section apply to all version of DDS protocol.

A client specifies which messages it wants to retrieve by sending a “Search Criteria” file to the server. Search Criteria file format is described in section 4.1.

The client SHOULD transmit the desired search criteria to the server at least once prior to retrieving data. In other words, the client SHOULD NOT make any assumptions about what search criteria (if any) are in effect on the server.

Once sent, a search criteria stays in effect for the duration of the session, or until another search criteria is sent.

The type-code for IdCriteria is ‘g’.

```
SearchCritReq ::= FiftyBlanks CriteriaText
FiftyBlanks ::= 50*( SP )     # 50 ASCII space characters
CriteriaText ::= OCTET_STRING
# Response is either error or just the 50-blanks
SearchCritResp ::= ErrorBody | FiftyBlanks
```

The “FiftyBlanks” field used to contain a file-name that was used only by the client (the server simply echoed it). This is deprecated. Clients and servers should fill this field with exactly 50 space characters.

The CriteriaText is a variable length text buffer containing the file contents.

Lines in the file-data MUST be terminated by a single line-feed character.

Search Criteria files MUST NOT be longer than 16000 bytes.

On success, the server MUST respond with valid message of type IdCriteria (‘g’). The message body SHOULD contain the 50 space characters only. Note that older servers echoed the complete search criteria file. Clients SHOULD be coded to allow (and ignore) this.

Upon receiving a search criteria file, the server must evaluate the criteria and establish a session context.

4.1 Search Criteria File Format

A search criteria file is a text file containing a series of keyword-value pairs, one per line, separated by a colon:

```
KEYWORD: Value
```

Each line begins with a keyword, followed by a colon, followed by a string value. Here are the available keywords:

DRS_SINCE	Only retrieve messages that were received by this system after the specified time. *See allowable time formats below.*
DRS_UNTIL	Only retrieve messages that were received by this system before the specified time. *See allowable time formats below.*
DAPS_SINCE	Only retrieve messages with a DAPS time-stamp after the specified time. *See allowable time formats below.*
DAPS_UNTIL	Only retrieve messages with a DAPS time-stamp before the specified time. *See allowable time formats below.*
NETWORK_LIST	The value following this keyword is a network list file. Only retrieve messages whose DCP address is contained in the list. For multiple lists, put multiple lines in the search criteria file, each beginning with the NETWORK_LIST keyword. Network list file names MUST NOT contain path components.
DCP_ADDRESS	Only retrieve messages with the specified DCP address. To specify multiple addresses, put multiple lines in the search criteria file, each beginning with the DCP_ADDRESS keyword.
DCP_NAME	Only retrieve messages with the specified DCP name. Names are mapped to DCP addresses in network list files. See the section below on Network List Files for details.
CHANNEL	Only retrieve messages that were transmitted on the specified GOES channel. The value is a number only. The GOES spacecraft identifier (‘E’ or ‘W’) is not necessary.

4.1.1 Allowable Time Formats for a Search Criteria File

The SINCE and UNTIL values can take one several time formats.

Relative formats start with the keyword “now” and then add or subtract increments. For example:

```
now – 20 minutes
now – 1 day
now – 1 week 3 days 20 minutes 10 seconds
now
```

You can specify an absolute GMT value in one of the following formats.

```
YYYY/DDD HH:MM:SS      complete specification
YYYY/DDD HH:MM         seconds assumed to be 00
DDD HH:MM:SS           assume current year
DDD HH:MM              seconds assumed to be 00
HH:MM:SS               assume current day
HH:MM                  seconds assumed to be 00
```

Output will stop when all messages currently on the server have been retrieved.

5 Transferring Network Lists to/from the Server

This capability exists in DDS protocol version 2 and higher. Clients MUST NOT send this type of request to version 1 servers.

DDS Version 2 added a new capability for transferring network list files to and from a server. Version 1 relied on network lists being persistently stored and maintained on the server.

Servers MAY implement a separate mechanism for storing network lists persistently. This relieves clients from the burden of sending them at the start of each session. Persistent lists can be stored and maintained securely, and are available for reference by any client.

Clients MAY upload *transient* network lists to the server at any time in a session. Network lists transferred via DDS SHOULD be considered transient lists. Clients SHOULD NOT make any assumption regarding what transient network lists reside on the server. In particular, a client SHOULD NOT assume that a list uploaded in a previous session is still available on the server.

The only size limitation imposed on a transient network list by the protocol is that it must fit in a single protocol-message. The header uses 5 digits to represent the message body length. This means that transient network lists are limited to (99999 – 64) bytes.

5.1 Sending a Transient Network List to the Server

The type-code for IdPutNetlist is ‘j’. This command uploads a list from the client to the server.

```
PutNetlistReq ::= filename ListText
filename ::= NAME { SP }    # Name left-justified in 64-char field
ListText ::= OCTET_STRING
# Response is either empty or an ErrorBody
PutNetlistResp ::= empty | ErrorBody
```

The file name field must be exactly 64 characters long. The name is left justified in the field and padded with blanks. The name field SHOULD NOT contain path separators ‘/’ or ‘\’. That is, it should be a simple filename.

On success, the server MUST respond with a DDS message with type IdPutNetlist (‘j’) with an empty message body.

Upon error, the server MUST respond with a message containing an ErrorBody with sufficient information for the client to diagnose the problem.

5.2 Retrieving Network Lists From the Server

Version 2 servers MUST implement this mechanism allowing clients to retrieve network lists. This applies to both transient lists and persistent lists.

The type-code for IdGetNetlist is ‘k’. This command downloads a list from the server to the client.

```
GetNetlistReq ::= filename

filename ::= NAME { SP }  # Name left-justified in a 64-char field

GetNetlistResp ::= ErrorBody | ( filename ListText )
```

Following the header is a 64-character field containing the network list file name, left justified. The name field SHOULD NOT contain path separators ‘/’ or ‘\’. That is, it should be a simple filename.

On success, the server MUST respond with a DDS message of type IdGetNetlist (‘k’) containing:

- a 64 character field containing the network list file name, left justified.
- a variable-length field containing the network list file contents.

If the specified list is not available on the server a response of type IdGetNetlist with an ErrorBody MUST be returned.

5.3 Network List File Format

Network List Files are ASCII Files containing a DCP addresses, one per line. Each line MUST be terminated by a single line-feed character. The format of each line is as follows:

```
NetlistFile ::= { NetlistLine }

NetlistLine ::= DcpAddress [ ':' NAME [ SP Description ] ] EOL

Description ::= STRING

EOL ::= LF | CRLF
```

Example

```
CE3E13BC:WTSM5 Chippewa River Diversion Dam near Watson, MN
CE3E86DE:GLKM5 GULL LAKE ELEVATION   near Brainerd, MN
CE456DFA:BIFM5 BIG FORK RIVER AT BIG FALLS, MN
CE45705E:GPOM5 LAKE KABETOGAMA AT GOLD PORTAGE, MN
CE457E8C:SSIM5 LAKE OF THE WOODS AT SPRING STEEL ISLAND, nr Warroad, MN
```

6 Retrieving Data

After connecting, authenticating, sending network lists, and sending search criteria, a client typically enters a loop where it continually polls for the next message that passes the criteria.

In both mechanisms described below for retrieving data, the server MUST only send DCP messages that match the client's search criteria.

All DCP Messages MUST start with the standard 37-byte DOMSAT header as defined in Table 6-1.

Offset	***Length***	***Type***	***Description***
0	8	hexstring	8 hex digit DCP address
8	11	time	Time formatted as YYDDDHHMMSS in UTC.
19	1	char	Message type codes 'G' means a good message, '?' means a message of questionable quality. Other type codes indicate DAPS-generated status messages.
20	2	integer	2 digit signal strength. Signal Strength will be two ASCII digits and will be in the range of 32 to 57. Signal strength is the implied EIRP, assuming the pilot is a +47 dBm reference.
22	2	sign digit	A + or - sign followed by a single digit or the letter 'A'. Represents frequency offset in units of 50 Hz. A represents the maximum offset of 500 Hz.
24	1	char	Modulation Index, coded as follows: **N** Normal: (60° ± 5°) **L** Low: (≤ 50°) **H** High: (≥ 70°)
25	1	char	Data Quality Indicator, coded as follows: **N** Normal: Error rate better than 10^{-6} **F** Fair: Error rate between 10^{-4} and 10^{-6} **P** Poor: Error rate worse than 10^{-4}
26	3	integer	3-digit GOES channel number, zero-filled.
29	1	char	GOES Spacecraft indicator (E or W)
30	2	hexstring	2 hex digits representing uplink carrier status.
32	5	integer	5-digit message length. This is the exact number of characters to follow.

Table 6-1: DOMSAT Header Contents.

6.1 Retrieving a Single Message per Request

This message type exists in all protocol versions.

To request a single DCP message, the client sends a DDS message of type IdDcp (‘f’) with an empty message body, and then waits for a response.

The server constructs a response message, again with type IdDcp (‘f’) followed by:

- A 40-character field containing a unique file-name that could be used to store this message on the client. This field is legacy from the original implementation.
- A variable length field containing the 37-byte DOMSAT Header followed by the DCP message.

6.1.1 Semantics for Until Time and Real-Time Retrieval

If the “until” time specified in the search criteria is reached, the server MUST respond with an error message with ServerCode DUNTIL (35).

If the search criteria contains no until time, this indicates that the client wishes to ‘hang on the line’, retrieving data in real-time as it becomes available. When the server receives an IdDcp request, AND no until time has been set, AND there are no new messages that meet the client’s criteria, THEN the server MUST respond with an error message with ServerCode = DMSGTIMEOUT (11). When the client receives this response, it SHOULD pause briefly and then try the request again.

6.2 Retrieving Multiple Messages per Request

This request type was added for protocol version 4. Clients MUST NOT send this request to servers that do not support protocol version 4.

To request multiple DCP messages per request, the client sends a DDS message of type IdDcpBlock (‘n’). The request has an empty (zero-length) body.

The server MUST send a response of type IdDcpBlock. The body of the response will be either an ErrorBody or it will contain multiple DCP messages, back-to-back:

```
# Request body is empty
MultDcpReqBody ::= empty
# Response contains DCP messages back-to-back:
MultDcpRespBody ::= ErrorBody | MultMessages
MultMessages ::= DcpMessage { DcpMessage }     # at least 1 message
DcpMessage ::= DOMSATHeader DcpMsgBody
# DOMSATHeader ::= 37-bytes as defined in table
DcpMsgBody ::= OCTET_STRING     # Actual message bytes
```

The server will place messages into the response up to a maximum of 50,000 bytes. The server MUST only place complete DCP messages into the response. If the next message does not fit, the server MUST return the response and await the next request.

The “until time” and “real time retrieval” semantics described above for single message transfers also applies to multiple message requests.

The server SHOULD NOT delay more than 55 seconds before returning a response to the client. Hence the server MAY return shorter than the maximum-size response if its search engine is taking a long time to find messages matching the search criteria.

The client MUST NOT interpret a less-than-maximum-size response as a sign that the server is finished.

7 Reference Implementation

A reference implementation of DDS is included in the LRGS (Local Readout Ground Station) code, developed by Ilex Engineering, Inc. The client software is 100% Java. The server software contains some native code and is written to run on a LRGS/DOMSAT receiver.

The LRGS software was written under contract to the USGS and USACE and is open-source.

To obtain a copy contact the U.S. Geological Survey, Water Resources Division or send an email to info@ilexeng.com.